A Scheduling Solution for Starbucks at the UW

Pranav Prabhakar

Yimo Shen

Tiffany Tian

The Starbucks locations at the University of Washington - Seattle pride themselves on employing students as baristas to gain valuable on campus employment. For years, the program assistant would spend up to two weeks each quarter manually scheduling employee shifts, wasting valuable time manually adjusting for the erratic availability of students while attempting to meet the minimum demand. The goal of this project is to create a schedule for Starbucks at the UW, which satisfies the minimum and maximum labor needs for all three locations while maximizing the number of baristas working during peak hours to decrease wait time for customers. We developed a model which reads in student availability as a matrix and outputs an optimized schedule while adhering to the given constraints. The runtime of our program was under 10 minutes, which is significantly less than the time spent creating a schedule currently, even if we factor in any manual adjustments that need to be made after the fact. To ensure longevity, we allowed for our model to be implemented with any unique set of student employees, ensuring that Starbucks at the UW can utilize this software to save time scheduling in future academic quarters.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### 1. STARBUCKS AT THE UW

Being situated near the birthplace of Starbucks, the University of Washington - Seattle campus boasts three Starbucks locations where students can enjoy their favorite beverage or meal. In order to keep these locations staffed, the program coordinator employs dozens of UW students to staff and keep these locations serving the hundreds of patrons per day. Unfortunately, student schedules are erratic, and their working hours are vastly different from each other. Currently, the program coordinator at Starbucks must set aside weeks at the beginning of each academic quarter to manually create student employee schedules. This has proven to be a challenging and time-consuming task, as each Starbucks location carries its own requirements as to working conditions and peak hours.

The ultimate goal of any business is to maximize profit, and in the case of Starbucks, this means having the maximum number of employees working at any given time allowed by space capacity, but especially during peak hours. By speaking to the program assistant managing employee schedules, Linna Bounxayavong, we quickly identified that the process of manually determining shifts was detrimental to the overall success of Starbucks for two key reasons. Firstly, the time that is spent by Ms. Bounxayavong analyzing and implementing student availability impacts the attention that she can provide to other crucial tasks required to keep a business running. Secondly, while manual scheduling may eventually yield a workable schedule, there is no guarantee that a person has created an optimal schedule maximizing location productivity and employee satisfaction. Therefore, we embarked on the task of deriving a solution that, given the parameters provided by students and Starbucks currently, is able to generate an optimized schedule in a fraction of the time that Ms. Bounxayavong currently spends.

### 2. SCHEDULING WITH UW HFS

Currently, all of the Housing and Food Services at the University of Washington schedule shifts manually - similar to Starbucks. When conducting some basic background research, we found that throughout campus, hiring managers need to expend a significant amount of man-hours to develop a schedule. In looking at solutions, we looked outside of student, on-campus jobs in order to gain some inspiration.

We found that Starbucks at the University of Washington is in a unique situation. The majority of brick-and-mortar businesses outside of campus have significantly less variability in their shift scheduling. For example, a UW student would submit their shift availability forms, and from there, the scheduling decisions are made. Most other businesses do not have to work around these erratic schedules, and instead, hire fewer employees for longer, more constant shifts.

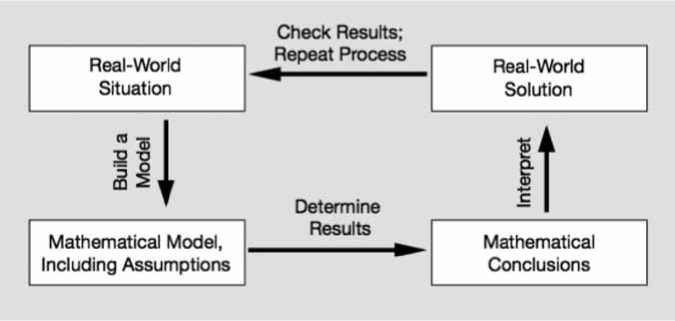
Therefore, in order to better understand the challenge that we are presented with, we will take you through the exact requirements that a hiring manager needs to consider when scheduling their employees’ shifts. First, each week is divided up, Monday through Friday, into 30-minute time blocks. A new employee would then mark the blocks they are not available to work, along with their preference for how many hours per week they would like to work. It is important to note that a student employee cannot work any more than 19.5 hours per week, due to legal restrictions.

Each of the three Starbucks locations has different hours of operation, peak hours, and barista capacity. *FYI: throughout this paper, the use of the word “barista” and “employee” will be used interchangeably.* We will describe the constraints in detail later in the paper, but the overview is as follows for each of the three locations\*:

* HUB
  + 9:00am - 3:00pm
  + 4 - 6 baristas
* POPULATION HEALTH
  + 7:00am - 5:00pm
  + 4 - 7 baristas
* SUZZALLO
  + 8:00am - 6:00pm
  + 4 - 10 barista

While the goal of our model is to ultimately maximize staffing subject to explicit constraints, each location has its respective peak hours, where it is imperative that we strive to achieve the maximum number of baristas working.

### 3. LINEAR OPTIMIZATION AND PuLP

Linear Optimization, or Linear Programming, is a method to achieve the “best” outcome of a mathematical model where the constraints and objectives are represented by linear equations. The “best” solution, as denoted earlier, is defined as the maximization or minimization of the objective function while still adhering to the determined constraints. In simpler terms, Linear Optimization allows us to tackle complex problems by simplifying them to a series of linear relationships. Applications of Linear Programming are present all around us. For example, when driving home from work, we would want to minimize the time spent commuting while adhering to constraints such as avoiding traffic and toll booths. Therefore, the best solution may not always be the shortest path home. We understand that many readers may be familiar with the mathematical modeling process. Therefore, we invite you to move on to Section 4: Our Approach. The remainder of this section will detail the four step process, as denoted in the figure below.

#### 3.1 Real World Situation

The first step to modeling is to determine our real world problem/goal. For us, we gathered data about the scheduling process at Starbucks, and determined that the current method is time consuming and perhaps not the optimal solution.

#### 3.2 Mathematical Problem

The next step, also known as translation, is to convert our real world problem into a mathematical goal. This involves first determining our decision variables. Decision variables are the unknown in an optimization problem, which will ultimately decide our output. We create our decision variables to write our objective function in order to maximize/minimize a quantity of interest. In our case, we chose to represent specific student working times and location as our decision variables in order to maximize the amount of students scheduled across all three locations at each of their peak hour times. Once our objective function is written, we can then determine our constraints that our model must adhere to. Constraints are represented as mathematical inequalities that signify real world limitations. One such constraint for us is that we are not able to have less than 4 and more than 10 baristas working at the Suzzallo location at a given time. Therefore, we would write that the amount of students scheduled at time *i* must be between 4 and 10, inclusive. Once our constraints and objective function are determined, we can work to solve our problem.

#### 3.3 Mathematical Solution and PuLP

Mathematical solving, known as prediction, can be done utilizing different frameworks. We decided to use PuLP, an open source linear programming package for Python which comes packaged with many industry standard solvers. We can pass in our mathematical functions into Python, and label them as part of a Linear Programming problem utilizing PuLP’s LpProblem method. Since the software has no background knowledge of our goal, we need to ensure that all constraints are determined, including lower/upper bounds, as well as return type (eg. Integers, continuous variables, etc.). Once our problem is formulated, we can use PuLP’s built in solver, which will choose the optimal solution method based on the format of our program.

In order to interpret our solution into usable data, some additional steps must be taken. PuLP will not only return the solution to our objective function, but also the coefficients of each variable used in our code. In our case, the solution to our objective function will be the total amount of students working during peak hours. This is not particularly useful to us, as we want to generate a schedule of working hours. However, by analyzing the status of our solution through PuLP, we are able to see the coefficients of our decision variables, which, as mentioned earlier, is the working hours and locations of each student passed through. From here, we are able to disregard the times where a student is not scheduled, and therefore generate a final schedule ready for use.

##### 3.3.1 Why PuLP?

As mentioned briefly prior, there are many Linear Optimization solvers that are available for use. We chose PuLP due to two key reasons. Firstly, it is easy to implement through Python’s framework, a language we are all familiar with. Furthermore, some other packages like Gurobi require matrix inputs for each of our decision variables, and data entry can become very time-consuming and prone to errors. PuLP, on the other hand, can take in our mathematical constraints as linear equations. Given that PuLP can comfortably handle the complexity of our problem, while mitigating potential errors and time spent, it was the sensible choice to achieve our goal.

#### 3.4 Real World Conclusions

Although our problem is now solved, there is still one last step in any modeling task, which is testing and analysis. Using our generated solution, we must compare it to previously determined solutions, in order to confirm that our objective is properly achieved. This is a crucial step in Linear Programming, as there is no point in generating a solution which does not optimize upon the original problem in any way. In our case, we use our schedule to ensure that student availability is better scheduled than the current method employed by Starbucks at the UW. Once we confirm this, we can then demonstrate the time saved and student shifts maximized by Linear Optimization as compared to Manual Scheduling.

### 4. SIMPLIFICATIONS

In order to effectively tackle this problem to the best of our ability, there were some simplifications and assumptions made on our end.

#### 4.1 Simplification One

For this attempt, we only received sample availability for 31 students. Therefore, we based our model on the availability of 31 student employees. Nevertheless, we should be able to effectively scale this model to any number of student employees, as our constraints do not take into account the total number of student employees accounted for.

#### 4.2 Simplification Two

Another simplification we had to make was regarding shift length. While, when speaking to Ms. Bounxayavong, it was clear that students worked either 3 or 3.5 hour shifts, for our draft, it made more sense to implement one of those two options. In a future iteration, we may work to include variable shift length, but for this model, we have fixed our shift lengths to 3.5 hours.

#### 4.3 Simplification Three

Finally, we accounted for full-time workers separately in our model. This is because their availability and shifts are fixed, and we do not need to schedule them. Instead, we used their predetermined schedules to ensure that we remained within the employee capacity. While this is not a simplification in itself, we did, however, not currently account for the 30 minute breaks that full time employees are allotted. This is because these breaks are flexible, and taken at the employee’s discretion.

This means our model would have some limitations. However, even if these smaller fixes need to be manually implemented, we still expect our platform to save a significant amount of time towards the generation of this schedule.

### 5. OUR APPROACH

#### 5.1 Sets and Variables

The goal of this project is to create a schedule for Starbucks at the UW, which satisfies the minimum and maximum labor needs for all three locations while maximizing the number of baristas during peak hours.

There were several ways we could have approached this problem. However, after analyzing and collecting the sheer amount of constraints, we realized that a Zero-One Integer Programming model is the best way to approach this problem. Zero-One Integer Programming [3] is a mathematical method using a series of binary functions utilizing “yes” and “no” answers to arrive at a conclusion. We began by listing out sets and creating variables shown below:

*Sets:*

* *Working days of week = {M, T, W, Th, F}*
* *Tday, shift , the set of all possible student shifts where day working days of week, and shift {1, 2,..., 28} represents 3.5-hour shifts where 1 = 4-7:30am\*, 2 = 4:30am-8am, … and 28 = 5:30-9pm.*
  + *\* NOTE: This is in order to avoid out-of-bounds exceptions when implementing in later constraints. The first 6 shifts and last 6 shifts are set to 0 to ensure no student will be scheduled at an invalid time.*
* *Tday, slot , the set of all possible time slots where day working days of week, and slot {1, 2,..., 28} represents 0.5-hour time slots throughout the day where 1 = 4-4:30am, 2 = 4:30am-5am, … and 28 = 5:30-6pm.*
  + *\* NOTE: This is in order to match with the starting time slot of each shifts. The first 6 time slots are set to 0.*
* *= {Availability of student barista i}.*
* *Z = {Suzzallo operating time slots: every working day for time* *slot* = *[9-28]},*
* *, peak hour time slots at Suzzallo.*
* *= {HUB operating time slots: every working day for time* *slot = [11-22]},*
* *, peak hour time slots at HUB.*
* *= {Pop Health operating time slots: every working day for time* *slot = [7-26]},*
* *, peak hour time slots at Pop Health.*

*Variables:*

For student baristas, we set as a binary decision variable:

where represents that student barista *i* is scheduled to work at the *jth* shift at location *k* and represents otherwise. Values of this decision variable are computed through Integer Programming methods in python using PuLP.

Similarly, for full time baristas, we define a binary variable

where represents that full-time barista *m* is working at the *nth* time slot at location *k* and represents otherwise. It is important to note that is predetermined based on the given schedules of full-time baristas.

#### 5.2 Objective

Given our goal of maximizing the number of employees working at any given time, we were able to derive the following objective function:

Where *F1* represents the total number of student baristas working throughout all three locations at a given time. Since full time baristas have fixed schedules, maximizing the number of student baristas will achieve the same goal of maximizing all baristas. Note that in order to calculate the total number of students working at a given time slot, we will be adding the number of students who started their shift 3 hours earlier, all the way until the students who started their shift at the beginning of this time slot, so mathematically, for each peak hour time slot t.

#### 5.3: Constraints

As mentioned, there are a series of constraints required to abide by in order to generate a schedule that can be implemented effectively. The given constraints and respective equations are detailed as follows.

##### 5.3.1 Barista Availability

The first constraint given concerns barista availability. As mentioned prior, each student barista submits their personal availability forms in order to be scheduled. Therefore, this constraint ensures that a barista is not scheduled during a time in which they are unavailable. We call this constraint C1. To represent this constraint mathematically, when a barista is not scheduled, :

##### 5.3.2 Space Constraints

While mathematically, it may be optimal to schedule a certain number of employees at a given location, there are some real-world limitations that tend to invalidate certain solutions. In our case, each location has both a maximum and minimum number of employees allowed to work at any given time. We have determined these constraints C2, C3, and C4, as follows:

*(C2) Suzzallo: 4-10 employees*

*(C3) HUB: 4-6 employees*

*(C4) Population Health: 4-7 employees*

##### 

##### 5.3.3 Operating Time Constraints

In addition to different levels of minimum and maximum number of employees, the three Starbucks locations also have different hours of operation. Student baristas can only be scheduled from within the hours of operation across all three locations. We have determined these constraints , , and , as follows:

() Suzzallo:

() HUB:

( Pop Health:

##### 5.3.4 One Shift Per day & Maximum Working Hours

The last constraint we need to implement is that a given employee can only work one shift per day. Without this implementation, our model may schedule a student multiple shifts back-to-back in a single day if their availability matrix deems them available, which is something that Starbucks would not prefer. This constraint will also indirectly satisfy the maximum working hours constraint for student baristas where Washington State Law does not allow a part-time student employee to work more than 19.5 hours per week. The one-shift-per-day condition will cap the maximum possible working hours for a student barista to be at 3.5 x 5 = 17.5 hours, as Starbucks only opens on the weekdays. Hence, constraint C8 would be as follows:

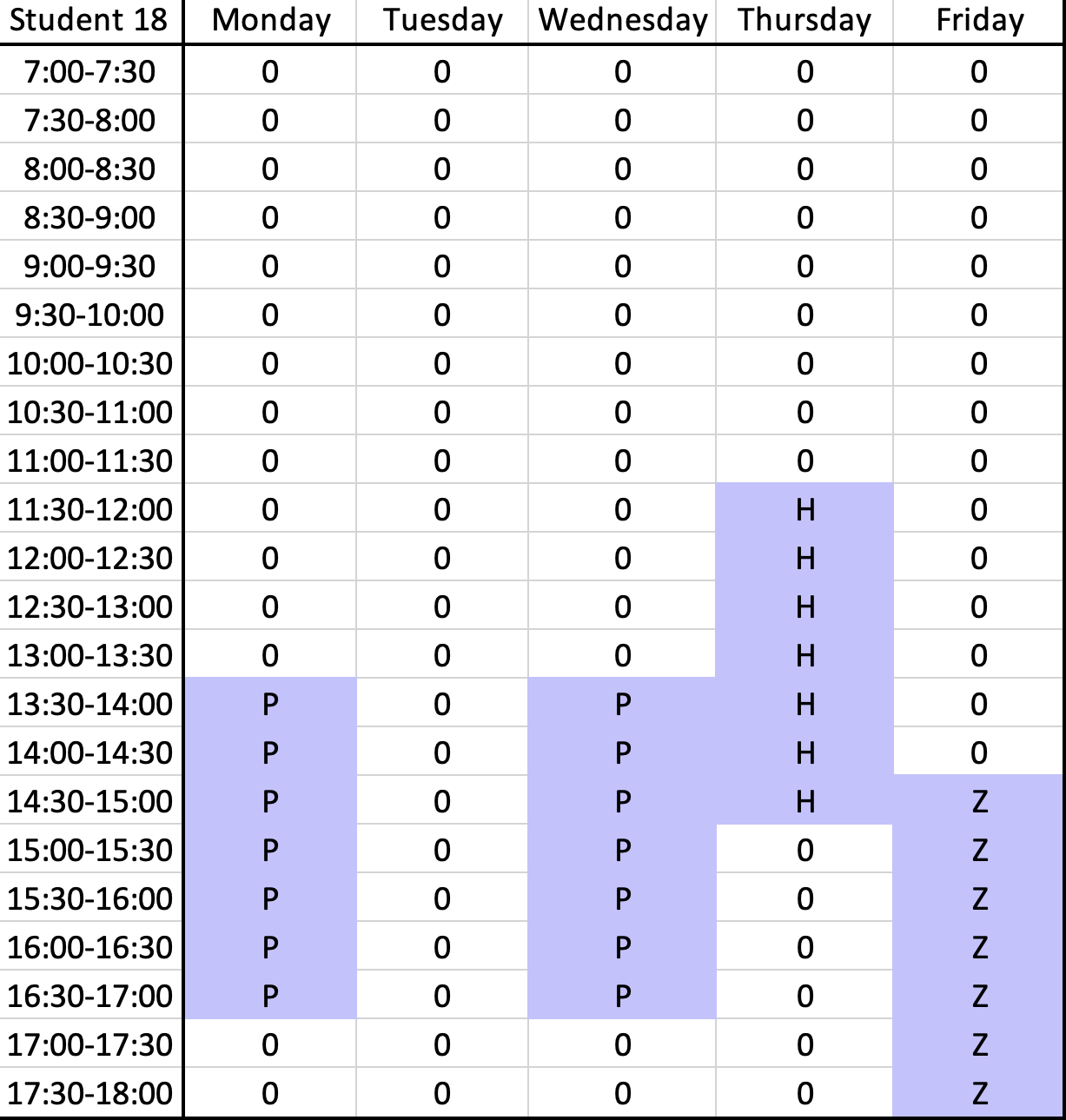
### 6. Solution

#### 6.1 Student Availability Forms

In order to gather the data for our student model, we needed to devise a method to parse through the student availability forms. Currently, each student availability form is provided as a table in Excel, with a cell for each half-hour shift every day of the week. The student then marks the cells in which they cannot work, and submits this sheet to Ms. Bounxayavong. Utilizing R, we created a simple method which reads in individual availability forms for each student, and outputs a single matrix containing all the availability. In this case, we generated a 31x140 matrix to input into our model, given 31 students and 140 time slots.

#### 6.2 Results

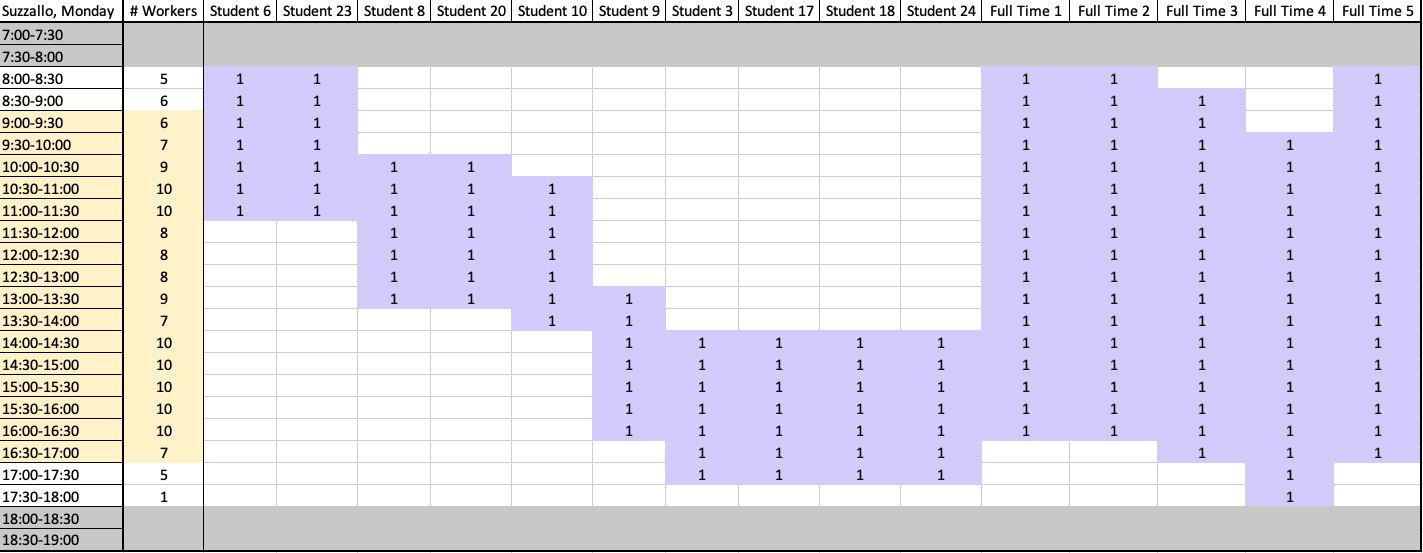
Once our PuLP problem had been adequately set up, we were able to run the solver and procure a feasible result. The solver currently outputs our code line-by-line. As we have 3 locations, 140 possible shifts, and 31 employee availability forms, this results in over 10,000 lines of code. However, by only printing the results in which our binary variable was equal to 1, we were able to generate the desired output to be then converted to actual schedules in Excel using R. The R code was able to generate not only a weekly schedule for each individual student (see Figure 2, which shows the sample schedule for Student #18 given their availability), but also a weekly schedule for each location that provides a comprehensive overview of who works at which location at what time (see Figure 3*)*.



#### 

*Figure 2*

As we can see, Figure 2 shows the weekly schedule of student 18. This particular student is scheduled to work multiple 3.5 hour shifts, where the location of their shift is denoted by the letter in each element of the table. Figure 3 shows the weekly schedule of the Suzzallo location on a Monday. This view is optimal for the program assistant, as they can easily reference the availability of all their employees on any given weekday. Although we were able to maximize staffing throughout peak hours, we found it impossible to fully staff all locations during peak hours given the current availability. Therefore, we recommend hiring additional students in order to meet the maximum capacity.

*Figure 3*

#### 6.3 Analysis

At this point in time, we do not have a current sample schedule for Starbucks @ UW to compare our schedule to. We aim to receive one soon, so that we can make comparisons in terms of student schedule optimization.

However, we are able to conclude that our generated schedule was created in a fraction of the time that it takes to formulate one manually. This will save the program assistant weeks every quarter, and will allow Starbucks to better use their resources elsewhere. We expect that the program assistant may have to make manual adjustments to our generated schedule due to extraneous variables, but with our output as a baseline, they already have a significant headstart to the schedule building process.

### 7. Future Improvements

There are several key factors that our group is excited to improve on, detailed as follows.

#### 7.1 Student Satisfaction

We are striving to implement a second goal of maximizing student satisfaction. Once we have an implemented, working linear optimization model for our current objective function, F1, we can look to add a second maximization criterion, and the related constraints, in order to take into account student location preferences. However, the obvious caveat here is that maximizing two things at once may not be feasible. This is because prioritizing two criteria may not result in an optimized solution for either equation. A potential method to counteract this could be to follow a similar strategy that Guido Cocchi, et al. detailed in their paper, *Scheduling the Italian National Volleyball Tournament* [2]. By creating a weighted variable, along with the relevant weighting for employee satisfaction, we could add to our original objective function to achieve both goals. This is above and beyond what the on-campus hiring process currently offers, however, it would be a great improvement for not only increasing employee satisfaction, but productivity as well, and therefore revenue.

#### 7.2 Staff Training Period

An assumption we made is to disregard the training period for a new staff member, as the training period for a new staff member is relatively short, and varies from employee to employee. In the future, we may look to provide an option to note if a recent employee is in training. This would not affect the schedule much, since it would just put a limitation on the number of new employees that can be present at a certain location at any given time.

7.3 PuLP Solver

The final potential improvement would be to examine the runtime of the default PuLP solver. By default, PuLP utilizes CBC, which is based on simplex algorithms, but incorporates other algorithms such as Branch-and-Cut, or Cut-generation. It is not necessary to detail how these solvers work, however, there is potential for timesave by passing our linear model into other algorithms. Given the time complexity of our solution is well within reasonable limits, this is not a priority for our problem. However, if more schedules are generated, or our solution is applied to a larger scale problem, it may be optimal to ensure that our runtime is minimal by examining the best solution process.

### 8. Acknowledgements

We would like to thank Linna Bounxayavong for taking the time out of her busy schedule to trust us to undertake this project. Without your help and willingness, we would not have the resources to undertake this complex task.

We would also like to thank our professor, Sara Billey. The structure of this class is unlike many other mathematical courses we have taken, however, it is thanks to this, and your enthusiasm for the material, that we remain as engaged and excited as we are.

### 9. References

[1] Kapuscinski et al.: Inventory Decisions in Dell’s Supply Chain. Interfaces 34(3), pp. 191–205, © 2004 INFORMS

[2] Guido Cocchi, Alessandro Galligari, Federica Picca Nicolino, Veronica Piccialli, Fabio Schoen, Marco Sciandrone (2018) Scheduling the Italian National Volleyball Tournament. Interfaces 48(3):271-284. <https://doi.org/10.1287/inte.2017.0932>

[3] Winston, Wayne L, and Jeffrey B. Goldberg. Operations Research: Applications and Algorithms. Belmont, CA: Thomson/Brooks/Cole, 2004. Print. Turabian (6th ed.).

### 10. Appendix

#### 10.1 R code for generating the availability matrix

```{r}

# transcribe the student availabilities from seperate excel files to a matrix

library(xlsx)

library(dplyr)

library(tidyverse)

number\_of\_students = 31

days\_of\_week = 5

daily\_time\_slots = 28

shift\_length = 7

weekly\_time\_slots = days\_of\_week \* daily\_time\_slots

daily\_shifts = daily\_time\_slots + shift\_length - 1

weekly\_shifts = days\_of\_week \* daily\_shifts

A = data.frame(rep(NA, weekly\_shifts))

# Loop through each and every student barista's availability

for (i in 1:number\_of\_students){

df <- read.xlsx("./Student Availabilities.xlsx", sheetIndex = i)[1:daily\_time\_slots,]

df = df[, 2:(days\_of\_week + 1)]

df = replace(df, !is.na(df), 0) # replace unavailable slots with 0

df = replace(df, is.na(df), 1) # replace available slots with 1

# Create data frame to store a person's weekly availability

C = data.frame(rep(NA, daily\_shifts))

# Loop through each day of the week

for (l in 1:days\_of\_week){

# the first 6 shifts (start time being 4am-6:30am) are scheduled to be empty

B = c(rep(0, shift\_length - 1))

# Loop through the rest of the start times at the (j+7)th slot

for (j in (1:(daily\_time\_slots - (shift\_length - 1)))){

availability = 0

for (k in 0:(shift\_length - 1)){

availability = availability + as.numeric(df[j + k, l])

}

# if the person is available throughout the entire next 3.5 hours

if (availability == shift\_length){

B[j + shift\_length - 1] = 1 # they are available with start time at the (j+7)th slot

} else {

B[j + shift\_length - 1] = 0

}

}

# the last 6 shifts (start time being 3pm-5:30pm) are scheduled to be empty

for (j in 1:(shift\_length - 1)){

B[daily\_time\_slots + j] = 0

}

# Store availability of that day in the lth column

C[l] = B

}

# Transform weekly availability data frame to a vector

A[i] = c(unlist(C))

}

colnames(A) = c(1:number\_of\_students)

write.xlsx(A, "./Student\_availability\_matrix.xlsx")

```

```{r}

# transcribe the full-time availabilities from seperate excel files to a dataframe

D = data.frame(matrix(0, nrow = weekly\_time\_slots, ncol = 3))

colnames(D) = c("HUB", "Suzzallo", "Pop.Health")

number\_of\_fulltime = 8

for (i in 1:number\_of\_fulltime) { #loop through each student barista

df\_ft = read.xlsx("./Full-time Availabilities.xlsx", sheetIndex = i)[1:daily\_time\_slots,]

location = colnames(df\_ft)[1]

df\_ft = lag(df\_ft, shift\_length - 1)

df\_ft = df\_ft[, 2:(days\_of\_week + 1)]

df\_ft = replace(df\_ft, !is.na(df\_ft), 1) # replace available slots with 1

df\_ft = replace(df\_ft, is.na(df\_ft), 0) # replace unavailable slots with 0

D[location] = unlist(D[location],use.names=FALSE) + as.numeric(unlist(df\_ft,use.names=FALSE))

}

write.xlsx(D, "./Fulltime\_availability\_df.xlsx")

```

#### 10.2 Python model for Linear Programming Solver

from pulp import LpMaximize, LpProblem, LpStatus, lpSum, LpVariable

import pandas as pd

import numpy as np

import sys

sys.setrecursionlimit(1000000)

# Create the model

model = LpProblem(name="model", sense=LpMaximize)

number\_of\_students = 31

number\_of\_fulltimes = 8

days\_of\_week = 5

daily\_time\_slots = 28

shift\_length = 7

weekly\_time\_slots = days\_of\_week \* daily\_time\_slots

daily\_shifts = daily\_time\_slots

weekly\_shifts = days\_of\_week \* daily\_shifts

# Initialize the decision variables

for i in range(1, number\_of\_students + 1):

for j in range(1, weekly\_shifts + 1):

for k in ["Z", "P", "H"]:

str = 'x\_{0}\_{1}\_{2} = LpVariable("x\_{3}\_{4}\_{5}", lowBound=0, upBound = 1, cat=\'Integer\')'.format(i,j,k,i,j,k)

exec(str)

# Define sets

# Create sets for hours of operations and peak hours of the week

Z = list()

Z\_peak = list()

H = list()

H\_peak = list()

P = list()

P\_peak = list()

Z\_start = 3 + 6

Z\_end = 23 + 6

Z\_peak\_start = 5 + 6

Z\_peak\_end = 21 + 6

H\_start = 5 + 6

H\_end = 17 + 6

H\_peak\_start = 9 + 6

H\_peak\_end = 13 + 6

P\_start = 1 + 6

P\_end = 21 + 6

P\_peak\_start = 7 + 6

P\_peak\_end =15 + 6

for day in range(days\_of\_week):

Z.extend(list(range(Z\_start + daily\_time\_slots \* day, Z\_end + daily\_time\_slots \* day)))

Z\_peak.extend(list(range(Z\_peak\_start + daily\_time\_slots \* day, Z\_peak\_end + daily\_time\_slots \* day)))

H.extend(list(range(H\_start + daily\_time\_slots \* day, H\_end + daily\_time\_slots \* day)))

H\_peak.extend(list(range(H\_peak\_start + daily\_time\_slots \* day, H\_peak\_end + daily\_time\_slots \* day)))

P.extend(list(range(P\_start + daily\_time\_slots \* day, P\_end + daily\_time\_slots \* day)))

P\_peak.extend(list(range(P\_peak\_start + daily\_time\_slots \* day, P\_peak\_end+daily\_time\_slots \* day)))

# Read in Student and Full time availability matrices

As = pd.read\_excel("./Student\_availability\_matrix.xlsx")

Af = pd.read\_excel("./Fulltime\_availability\_df.xlsx")

# Add the constraints to the model

# C1

for i in range(1, number\_of\_students + 1): # loop through all student baristas

for j in range(1, weekly\_shifts + 1): # loop through all the possible shifts

for k in ["Z", "H", "P"]:

if As.loc[j-1][i] == 0:

C\_1 = 'x\_{0}\_{1}\_{2}'.format(i,j,k)

model += (eval(C\_1) <= 0, 'C1 constraint for student {0} at time slot {1} location {2}'.format(i,j,k))

# C2

for t in Z: # at each operation time slot t

# initialize the constraint function

C\_2 = ""

# add all students who are working at that time slot t

for i in range(7, number\_of\_students + 1):

# NOTE: this will add everyone who started working up to 6 time slots before that time slot

for l in range(0, shift\_length):

C\_2 += 'x\_{0}\_{1}\_Z + '.format(i,t - l)

# add all full-times who are working at that time slot

for m in range(1, number\_of\_fulltimes + 1):

f\_z = Af["Suzzallo"][t-1] # number of full-time working at Suzzallo at time slot t

C\_2 = C\_2[:-2] # remove the last plus sign

# Add constraints for time slot t into the model

model += (eval(C\_2) <= 10 - f\_z,

'C2 upperbound constraint for time slot {0} at Suzzallo'.format(t))

model += (eval(C\_2) >= 5 - f\_z,

'C2 lowerbound constraint for time slot {0} at Suzzallo'.format(t))

# C3

for t in H: # at each operation time slot t

# initialize the constraint function

C\_3 = ""

# add all students who are working at time slot t

for i in range(7, number\_of\_students + 1):

# NOTE: this will add everyone who started working up to 6 time slots before time slot t

for l in range(0, shift\_length):

C\_3 += 'x\_{0}\_{1}\_H + '.format(i,t - l)

# add all full-times who are working at that time slot

for m in range(1, number\_of\_fulltimes + 1):

f\_h = Af["HUB"][t-1] # number of full-time working at Suzzallo at time slot t

C\_3 = C\_3[:-2] # remove the last plus sign

# Add constraints for time slot t into the model

model += (eval(C\_3) <= 6 - f\_h,

'C3 upperbound constraint for time slot {0} at HUB'.format(t))

model += (eval(C\_3) >= 4 - f\_h,

'C3 lowerbound constraint for time slot {0} at HUB'.format(t))

# C4

for t in P: # at each operation time slot t

# initialize the constraint function

C\_4 = ""

# add all students who are working at time slot t

for i in range(1, number\_of\_students + 1):

# NOTE: this will add everyone who started working up to 6 time slots before time slot t

for l in range(0, shift\_length):

C\_4 += 'x\_{0}\_{1}\_P + '.format(i,t - l)

# add all full-times who are working at that time slot

for m in range(1, number\_of\_fulltimes + 1):

f\_p = Af["Pop.Health"][t-1] # number of full-time working at Suzzallo time slot t

C\_4 = C\_4[:-2] # remove the last plus sign

# Add constraints for time slot t into the model

model += (eval(C\_4) <= 7 - f\_p,

'C4 upperbound constraint for time slot {0} at Pop Health'.format(t))

model += (eval(C\_4) >= 4 - f\_p,

'C4 lowerbound constraint for time slot {0} at Pop Health'.format(t))

# C5

# Loop through each student

for i in range(1, number\_of\_students + 1):

# Loop through all time slots

for time in range(7, weekly\_time\_slots + 1):

C\_5 = ""

# If it is not within the operation hours

if time not in Z:

for l in range(0, shift\_length):

# there should not be shift scheduled at this location

C\_5 += 'x\_{0}\_{1}\_Z +'.format(i, time - l)

C\_5 = C\_5[:-2] # remove the last plus sign

model += (eval(C\_5) <= 0, 'C5 constraint for student {0} for time slot {1}'.format(i, time))

else:

pass

# C6

# Loop through each student

for i in range(1, number\_of\_students + 1):

# Loop through all time slots

for time in range(7, weekly\_time\_slots + 1):

C\_6 = ""

# If it is not within the operation hours

if time not in H:

for l in range(0, shift\_length):

# there should not be shift scheduled at this location

C\_6 += 'x\_{0}\_{1}\_H +'.format(i, time - l)

C\_6 = C\_6[:-2] # remove the last plus sign

model += (eval(C\_6) <= 0, 'C6 constraint for student {0} for time slot {1}'.format(i, time))

else:

pass

# C7

# Loop through each student

for i in range(1, number\_of\_students + 1):

# Loop through all time slots

for time in range(7, weekly\_time\_slots + 1):

C\_7 = ""

# If it is not within the operation hours

if time not in P:

for l in range(0, shift\_length):

# there should not be shift scheduled at this location

C\_7 += 'x\_{0}\_{1}\_P +'.format(i, time - l)

C\_7 = C\_7[:-2] # remove the last plus sign

model += (eval(C\_7) <= 0, 'C7 constraint for student {0} for time slot {1}'.format(i, time))

else:

pass

# C8

# Loop through each student

for i in range(1, number\_of\_students + 1):

# For every day

for day in range(days\_of\_week):

C\_8 = ""

# Add all the shifts they have that day

for t in range(1, daily\_shifts + 1):

# in each location

for k in ["Z", "H", "P"]:

C\_8 += 'x\_{0}\_{1}\_{2} +'.format(i, day \* daily\_shifts + t, k)

C\_8 = C\_8[:-2] # remove the last plus sign

model += (eval(C\_8) <= 1, 'C8 constraint at day {0} for student {1}'.format(day + 1,i))

# Define objective function (F1)

F\_1 = ""

for i in range(1, number\_of\_students + 1):

for j in Z\_peak:

for l in range(0, shift\_length):

F\_1 += 'x\_{0}\_{1}\_Z + '.format(i,j-l)

for j in H\_peak:

for l in range(0, shift\_length):

F\_1 += 'x\_{0}\_{1}\_H + '.format(i,j-l)

for j in P\_peak:

for l in range(0, shift\_length):

F\_1 += 'x\_{0}\_{1}\_P + '.format(i,j-l)

F\_1 = F\_1[:-2] # remove the last plus sign

model += eval(F\_1)

status = model.solve()

print(f"status: {model.status}, {LpStatus[model.status]}")

print(f"objective: {model.objective.value()}")

for var in model.variables():

print(f"{var.name}: {var.value()}")

for name, constraint in model.constraints.items():

print(f"{name}: {constraint.value()}")

output = []

for var in model.variables():

if var.value() == 1:

output.append(var.name)

df = pd.DataFrame(columns = ['Student', 'Shift Time', 'Location'])

for line in output:

df = df.append({'Student' : line.split('\_')[1], 'Shift Time' : line.split('\_')[2], 'Location' : line.split('\_')[3]},

ignore\_index = True)

df.to\_excel(r"./output.xlsx")

#### 10.3 R Code for Converting Output into Schedules

```{r}

# list of constant variables

number\_of\_students = 34

days\_of\_week = 5

daily\_time\_slots = 28

shift\_length = 7

weekly\_time\_slots = days\_of\_week \* daily\_time\_slots

daily\_shifts = daily\_time\_slots + shift\_length - 1

weekly\_shifts = days\_of\_week \* daily\_shifts

weekdays = c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")

# shift to time slot function

shift\_to\_timeslot <- function(shift) {

sort(c(shift:(shift-shift\_length + 1)))

}

```

```{r}

# output schedule for each student

output <- read.xlsx("./output.xlsx", sheetIndex = 1)[,2:4]

output$Student = as.numeric(output$Student)

output$Shift.Time = as.numeric(output$Shift.Time)

output\_per\_student = split(output, output$Student)

for (i in 1:length(output\_per\_student)) { # loop through output for each student

# compute list of shift times for each student

shifts = output\_per\_student[i][[1]]$Shift.Time

locations = output\_per\_student[i][[1]]$Location

individual\_schedule = data.frame(matrix(0, nrow = daily\_time\_slots - shift\_length + 1, ncol = days\_of\_week))

colnames(individual\_schedule) = c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")

rownames(individual\_schedule) = c("7:00-7:30", "7:30-8:00", "8:00-8:30", "8:30-9:00", "9:00-9:30",

"9:30-10:00", "10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00",

"12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30",

"14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00",

"17:00-17:30", "17:30-18:00")

for (j in 1:length(shifts)) { # loop through each shift

shift = shifts[j]

location = locations[j]

day = floor(shift / daily\_time\_slots) + 1

shift\_daily = shift %% daily\_time\_slots

for (time in shift\_to\_timeslot(shift\_daily)){

individual\_schedule[time,day] = location

}

}

write.xlsx(individual\_schedule, file = "student\_schedule.xlsx", sheetName = as.character(i), append = TRUE)

}

```

```{r}

# output schedule for each day at Suzzallo

output\_per\_location = split(output, output$Location)

output\_suzzallo = output\_per\_location$Z

output\_suzzallo["Day"] = floor(output\_suzzallo$Shift.Time / daily\_time\_slots) + 1

output\_suzzallo\_per\_day = split(output\_suzzallo, output\_suzzallo$Day)

for (i in 1:length(output\_suzzallo\_per\_day)) { # loop through output for each day at Suzzallo

# compute list of shift times for each student

shifts = output\_suzzallo\_per\_day[i][[1]]$Shift.Time

students = output\_suzzallo\_per\_day[i][[1]]$Student

one\_day\_schedule = data.frame(matrix(0, nrow = daily\_time\_slots - shift\_length + 1, ncol = length(students)))

colnames(one\_day\_schedule) = students

rownames(one\_day\_schedule) = c("7:00-7:30", "7:30-8:00", "8:00-8:30", "8:30-9:00", "9:00-9:30","9:30-10:00", "10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00",

"12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00", "17:00-17:30", "17:30-18:00")

for (j in 1:length(students)) { # loop through each student

shift = shifts[j]

student = students[j]

shift\_daily = shift %% daily\_time\_slots

for (time in shift\_to\_timeslot(shift\_daily)){

one\_day\_schedule[time, j] = 1

}

}

one\_day\_schedule["Total Workers"] = rowSums(one\_day\_schedule)

write.xlsx(one\_day\_schedule, file = "suzzallo\_schedule.xlsx", sheetName = as.character(weekdays[i]), append = TRUE)

}

```

```{r}

# output schedule for HUB

output\_hub = output\_per\_location$H

output\_hub["Day"] = floor(output\_hub$Shift.Time / daily\_time\_slots) + 1

output\_hub\_per\_day = split(output\_hub, output\_hub$Day)

for (i in 1:length(output\_hub\_per\_day)) { # loop through output for each day at Suzzallo

# compute list of shift times for each student

shifts = output\_hub\_per\_day[i][[1]]$Shift.Time

students = output\_hub\_per\_day[i][[1]]$Student

one\_day\_schedule = data.frame(matrix(0, nrow = daily\_time\_slots - shift\_length + 1, ncol = length(students)))

colnames(one\_day\_schedule) = students

rownames(one\_day\_schedule) = c("7:00-7:30", "7:30-8:00", "8:00-8:30", "8:30-9:00", "9:00-9:30", "9:30-10:00", "10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00", "12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00", "17:00-17:30", "17:30-18:00")

for (j in 1:length(students)) { # loop through each student

shift = shifts[j]

student = students[j]

shift\_daily = shift %% daily\_time\_slots

for (time in shift\_to\_timeslot(shift\_daily)){

one\_day\_schedule[time, j] = 1

}

}

one\_day\_schedule["Total Workers"] = rowSums(one\_day\_schedule)

write.xlsx(one\_day\_schedule, file = "hub\_schedule.xlsx", sheetName = as.character(weekdays[i]), append = TRUE)

}

```

```{r}

# output schedule for Pop Health

output\_pop = output\_per\_location$P

output\_pop["Day"] = floor(output\_pop$Shift.Time / daily\_time\_slots) + 1

output\_pop\_per\_day = split(output\_pop, output\_pop$Day)

for (i in 1:length(output\_pop\_per\_day)) { # loop through output for each day at Suzzallo

# compute list of shift times for each student

shifts = output\_pop\_per\_day[i][[1]]$Shift.Time

students = output\_pop\_per\_day[i][[1]]$Student

one\_day\_schedule = data.frame(matrix(0, nrow = daily\_time\_slots - shift\_length + 1, ncol = length(students)))

colnames(one\_day\_schedule) = students

rownames(one\_day\_schedule) = c("7:00-7:30", "7:30-8:00", "8:00-8:30", "8:30-9:00", "9:00-9:30", "9:30-10:00", "10:00-10:30", "10:30-11:00", "11:00-11:30", "11:30-12:00",

"12:00-12:30", "12:30-13:00", "13:00-13:30", "13:30-14:00", "14:00-14:30", "14:30-15:00", "15:00-15:30", "15:30-16:00", "16:00-16:30", "16:30-17:00", "17:00-17:30", "17:30-18:00")

for (j in 1:length(students)) { # loop through each student

shift = shifts[j]

student = students[j]

shift\_daily = shift %% daily\_time\_slots

for (time in shift\_to\_timeslot(shift\_daily)){

one\_day\_schedule[time, j] = 1

}

}

one\_day\_schedule["Total Workers"] = rowSums(one\_day\_schedule)

write.xlsx(one\_day\_schedule, file = "pop\_health\_schedule.xlsx", sheetName = as.character(weekdays[i]), append = TRUE)

}

```

#### 